SPASE Query Language (SPASEQL)

http://spaseql.gsfc.nasa.gov

Description and Rationale

The Heliophysics data environment (HPDE) is comprised of several domain specific Virtual Observatories (VxOs). Each VxO focuses its efforts on the types of data and research unique to its domain. However, many pertinent research questions demand resources from multiple domains. As such, it would be beneficial for the HPDE to implement a means for the VxOs, as well as added-value services, to communicate with each other in a standardized manner. This is the goal of the SPASE Query Language (SPASEQL).

SPASEQL is based on the SPASE metadata model. SPASE has become an integral part of the HPDE, and is the basis for most descriptions of data resources. The intent is to carry this familiarity over into developing a standardized means of asking and answering questions. SPASEQL is simply the mechanism for standardizing the messaging between VxOs; it is implementation neutral. The query language does not dictate how queries are to be executed. This means that the VxOs use their domain expertise and existing infrastructure to execute queries. SPASEQL provides a front-end to each VxO that allows them to speak the same language to each other.

The number of VxOs is not overwhelming and the point can be made for learning the unique interfaces of each rather than developing a standardized language. However, this has several drawbacks. The resulting software libraries are dependent on the VxO interfaces remaining static. A change to an interface breaks communications. SPASEQL is a static front end to a VxO. The underlying VxO can change, as mandated by its community, however, the interface other VxOs see remains unchanged.

A second concern is the diversity and complexity of underlying data resources. This is evidenced in the ongoing evolution of SPASE. Thus, for each search capability a VxO implements (granules, spacecraft positions, time periods, etc.), it must have a corresponding means of formatting and packaging the results (the results message structure) for its API. As a result, interacting with VxO APIs scales not with the number of VxOs but rather with the total number of search capabilities. If left to develop independently, similar search capabilities at multiple VxOs will undoubtedly be returned via varying means. Thus, a granule search at one VxO may not be directly interoperable with a granule search at another VxO. This makes the integration of multiple VxO results a formidable challenge.

Rather than each VxO developing independent standards for creating response messages it makes sense to leverage the existing SPASE model. This model already contains agreed upon terminology and definitions that have resulted from years of hard work. SPASE already serves as common parlance for data descriptions. This capability can be extended into a common language for VxOs who are discussing these data descriptions. This makes the usage of the HPDE, as a collective whole, a much easier task.

Costs, Ease of Use, and Benefits

SPASEQL can lead to rapid development and reduced costs. Having a standardized communications language means that a question can be asked the same way from one domain to the next. As a result, software can be reused and queries can be saved and re-executed when needed. This leads to easier integration of VxOs and services. It also allows for the potential chaining of VxOs. That is, if we are speaking the same language, the results of one VxO query can easily be transformed into the constraints of a query to another VxO. Overall, SPASEQL should lead to reduced costs, reuse, and easier integration of the HPDE components.

Implementing SPASEQL does involve some costs. SPASEQL is a front-end that standardizes communications. The underlying VxO must be able to transform SPASEQL queries to execute on its underlying systems. The cost of this effort varies from VxO to VxO and from service to service. It is a function of the complexity and capabilities of the VxO or service. For example, VxOs implementing SQL databases (e.g. VITMO, VHO, and VMO/G) may find it difficult to transform complex SPASEQL queries into meaningful SQL queries. On the other hand, a simple plotting service can only accept very specific SPASEQL queries and may find it trivial to implement.

Additional costs come in the form of retrofitting existing services and VxO APIs, such as VSO who has a fully functioning API and predates SPASE. While SPASEQL can be used for such sites their owners will need to weigh the time and costs in retrofitting existing infrastructure.

Scope

SPASEQL will not handle all components of the HPDE. There will exist things outside the purview of SPASE, and ultimately SPASEQL. Some functionality will have to be implemented by other means. However, SPASEQL does address the basic set of questions that VxOs and services need to ask each other. As a first order interoperability agent it can go a long way in uniting the capabilities of the HPDE. VxO provided science use cases will help in defining the scope of SPASEQL.

Some possible use cases that have emerged from the HDMC are:

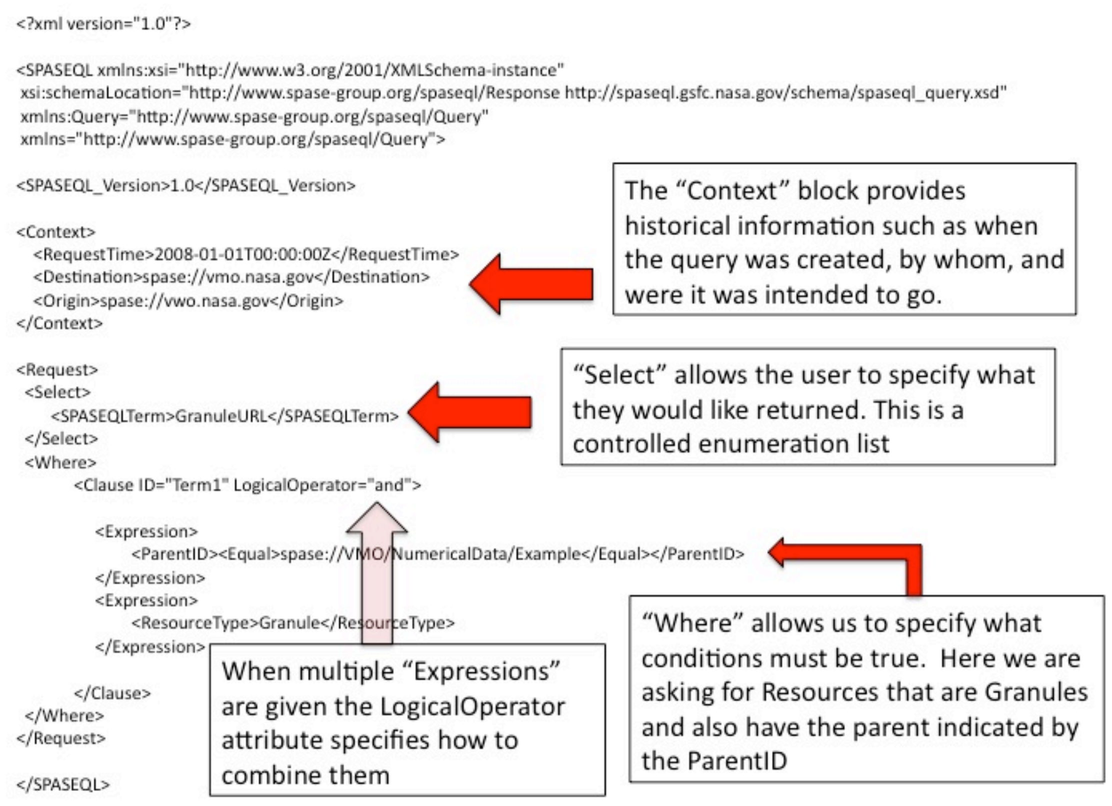
VSPO will serve as a global repository of resources. SPASEQL can serve as the means for VSPO to harvest new/modified metadata from the VxOs. In a similar fashion, VSPO can provide a SPASEQL interface by which others can collect and browse the metadata of specific VxOs.

VxO to VxO communication in order to answer user queries that span multiple domains. Each VxO aggregating information about other domains is redundant and wastes resources. SPASEQL will allow VxOs to take advantage of other domain VxOs’ expertise

Providing a standard means for VxOs to communicate with added-value services.

Anatomy of Query and Response Messages

The Query



```xml
<?xml version="1.0"?>

<SPASEQL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.spase-group.org/spaseql/Response http://spaseql.gsfc.nasa.gov/schema/spaseql_query.xsd"
 xmlns:Query="http://www.spase-group.org/spaseql/Query"
 xmlns="http://www.spase-group.org/spaseql/Query">

<SPASEQL_Version>1.0</SPASEQL_Version>

<Context>
   <RequestTime>2008-01-01T00:00:00Z</RequestTime>
   <Destination>spase://vmo.nasa.gov</Destination>
   <Origin>spase://vwo.nasa.gov</Origin>
</Context>

<Request>
 <Select>
     <SPASEQLTerm>GranuleURL</SPASEQLTerm>
 </Select>
 <Where>
        <Clause ID="Term1" LogicalOperator="and">

           <Expression>
                <ParentID><Equal>spase://VMO/NumericalData/Example</Equal></ParentID>
           </Expression>
           <Expression>
                <ResourceType>Granule</ResourceType>
           </Expression>

        </Clause>
 </Where>
</Request>

</SPASEQL>
```

The “Context” block provides historical information such as when the query was created, by whom, and were it was intended to go.

“Select” allows the user to specify what they would like returned. This is a controlled enumeration list

When multiple “Expressions” are given the LogicalOperator attribute specifies how to combine them

“Where” allows us to specify what conditions must be true. Here we are asking for Resources that are Granules and also have the parent indicated by the ParentID

Queries are expressed in XML and are of the form: SELECT X WHEN Y is true. They are somewhat analogous to SQL queries. The query has two parts – Context and Request. Context contains general information about when the query was submitted, who submitted it, and where it was intended to go. This section is intended for logging and archiving of queries.

The Request section contains the actual query request. The <Select> tag indicates what the user would like returned. In this case, the user is asking for Granule URLs. Servers may not be able to answer all possible <SPASEQLTerm> values. A SPASE Service description should be available for each service and this file will describe the capabilities of the service and which values it accepts. It should be noted that multiple <SPASEQLTerm> elements are allowed. Thus, a user can ask for any number of return items.

The <Where> element contains the restriction. This query states that the user would like Granule URLs when two conditions are true. In this case the user would like the Granule URLs for the NumericalData Resource with resourceID=spase://VMO/NumericalData/Example

This particular query contains one Clause element. The SPASEQL schema does allow complex queries to be formed by using multiple Clause elements. Consult the schema for more details.

The ambiguity of “and”

When constructing queries with multiple Clauses simply joining Clauses with “and” can be ambiguous. “And” does not specify enough information. The user could want the conditions to occur at the same time. Instead, the user could be asking that the conditions occur not only at the same time, but also within the same data product. In order to remove this ambiguity SPASEQL allows the attribute “conditionsOccurAt” with values "atSameTimeInSameProduct" and “atSameTime". Developers interested in complex queries should consult the schema for proper usage of this attribute.

The Response

```xml
<?xml version="1.0"?>

<SPASEQL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.spase-group.org/spaseql/Response http://spaseql.gsfc.nasa.gov/schema/spaseql_response.xsd"
 xmlns:Query="http://www.spase-group.org/spaseql/Query"
 xmlns="http://www.spase-group.org/spaseql/Response">

<ResponseContext>
        <QueryReceived>2008-01-01T00:00:00Z</QueryReceived>
        <ExecutionDuration>PT1M</ExecutionDuration>
        <ExecutedBy>spase://vho.nasa.gov</ExecutedBy>
        <QueryOrigin>spase://vwo.nasa.gov</QueryOrigin>
</ResponseContext>
```

This tells the user when the service received the query and how long it took to execute. It is purely informational for logging.

```xml
<Query>
<Query:Request>
    <Query:Select>
  <Query:SPASEQLTerm>GranuleURL</Query:SPASEQLTerm>
    </Query:Select>
    <Query:Where>
  <Query:Clause ID="Term1" LogicalOperator="and">
 <Query:Expression>
<Query:ParentID><Query:Equal>spase://VMO/NumericalData/Example</Query:Equal></Query:ParentID>
 </Query:Expression>
 <Query:Expression>
<Query:ResourceType>Granule</Query:ResourceType>
 </Query:Expression>
  </Query:Clause>
    </Query:Where>
</Query:Request>
</Query>
```

The original query is repeated in the response. The SPASEQL software library takes care of storing the query and rewriting it in the response. Services can exploit this feature of the reusable library.

```xml
<ResultSet>

<Result>
<GranuleURL>http://url/to/some/file/1</GranuleURL>
</Result>
<Result>
<GranuleURL>http://url/to/some/file/2</GranuleURL>
</Result>
<Result>
<GranuleURL>http://url/to/some/file/3</GranuleURL>
</Result>
<Result>
<GranuleURL>http://url/to/some/file/4</GranuleURL>
</Result>
<Result>
<GranuleURL>http://url/to/some/file/5</GranuleURL>
</Result>

</ResultSet>

</SPASEQL>
```

The results portion of the response. The user asked for granule URLs which are provided in the “ResultSet”. The values that can be asked to be returned are controlled by the SPASEQL schema

The response message has three parts. First, there is a similar context section detailing the execution of the query. Second, the query is repeated. This is so that users who wish to save and reuse queries have all the needed information in one message. The user does not need to save the query and response as a pair. For convenience the response message contains the results and the original query. The SPASEQL software library takes care of storing and rewriting the original query. Thus, this feature does not need to be reinvented by each service.

Finally, the <ResultsSet> element contains the results. Each result is placed in a <Result> element using a predefined name from the SPASEQL enumeration list.

Configuration

SPASE is a large and complex data model. Clearly, the VxO and services using SPASEQL will not be able to implement all of it. To accommodate this we created a configuration file based on the SPASE Service description. An example of this configuration can be found at http://spaseql.gsfc.nasa.gov/services/ This configuration file is compared against incoming queries to determine if the service is capable of answering the query.

Within the configuration file the "SelectOption" tells the service, and the user, what values can be used in the "SPASEQLTerm" portion of the query. The allowed values are: TimeSpan, GranuleURL, ProductID, GranuleID, InstrumentID, ObservatoryID, and MetadataURL. If TimeSpan is selected then the response "Result"s must have "StartDate" and "StopDate" elements.

Assumptions built into the SPASEQL language

Two SPASEQL attributes are optional. There are cases when they attributes are needed but are not supplied by the user. In such cases, services need to make assumptions. Built into the SPASEQL language are the following assumptions:

1.) if "conditionsOccur" is not specified a default value of "sameProductAtSameTime" will be used by services.

2.) If "LogicalOperator" attribute of "Where" is need and not supplied then services will assume "and".

Java Implementation

As each VxO and service will implement queries in their own particular way SPASEQL is implementation neutral. What we can provide are tools to send, receive, validate and assist in response creation. The SPASEQL Java code accomplishes this. We have attempted to assist as much as possible, however, each service will need to carry out the execution on their own.

In terms of query transmission we are currently using HTTP POST and assuming queries are written to XML once received by the server.

ServiceFunctions.java in the util/ package provides basic service functionality that can be reused by new services:

```
// Object that performs basic service features
ServiceFunctions service = new ServiceFunctions ();
SPASEQuery query = service.validate(schema, queryFile,
   config, responseFile);
Context context = service.createContext( serviceID,
  responseFile, query );
CreateResponse response = new CreateResponse ();
boolean errors = service.hasErrors(query);
```

In this example, we create a query object and validate the incoming query against both our service description (to see if we are able to answer it) and the SPASEQL schema (to ensure that it is valid SPASEQL). The reusable code then fills in the Context portion of the response message when we provide our serviceID and query. Finally, we check for errors during the validation before proceeding. Other reusable features of the SPASEQL library (not illustrated here) are a results parser and code to assist in creating the resultSet – see the response package.

Because SPASEQL allows complex subqueries, such as

((A and B) or (C and D))

the query is returned as a list of lists. We can access the lists with:

```
          System.out.println("The input query was...");
                // access to the original query
          System.out.println(query.xmlQuery);
          System.out.println();

        // loop through query components and execute them
       for (int i=0; i<query.numClauses; i++) {

         // Remove the next query object
                Object o = query.clauses.removeFromFront();
                // Determine if this is a complexClause or just a clause
                if ( o instanceof Clause ) {
                        Clause currentClause = (Clause) o;
                                while ( !clause.expressions.isEmpty() ) {
Expression expression = (Expression) clause.expressions.removeFromFront();

                        String[] constraints = constraint.getConstraint(expression);
                        System.out.println(expression.name + " " +
       Constraints[0] + " " + constraints[1] + " inclusive=" +
                                        constraints[2]);
                }
        }
```

Service Creation Guidelines

It is encouraged that service creators provide both command line and object execution methods for their services. This ensures wider usage of the service as users can execute the service from the command line or use the Java objects to incorporate the service into their own code. Source code examples for existing services can be found at http://spaseql.gsfc.nasa.gov/services/